INRIA

UNITÉ DE RECHERCHE
INRIA-RENNES

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P 105
78153 Le Chesnay Cedex
France
Tél. (1) 39 63 55 11

Rapports de Recherche

N° 1093

Programme 2
Structures Nouvelles d'Ordinateurs

THE MAPPING OF LINEAR RECURRENCE EQUATIONS ON REGULAR ARRAYS

Patrice QUINTON
Vincent Van DONGEN

Septembre 1989

RR_1093

IRISA INSTITUT DE RECHERCHE EN INFORMATIQUE
ET SYSTÈMES ALÉATOIRES

Campus Universitaire de Beaulieu
35042-RENNES CÉDEX
FRANCE
Téléphone: 99 36 20 00
Télex: UNIRISA 950 473 F
Télécopie: 99 38 38 32

The Mapping of Linear Recurrence Equations on Regular Arrays*
Mise en œuvre d'équations récurrentes linéaires sur des architectures régulières

Patrice Quinton†
Vincent Van Dongen‡

Publication Interne n° 485 - Juillet 1989 - 40 Pages

Résumé La parallélisation d'un grand nombre d'algorithmes peut être obtenue par des transformations spatio-temporelles appliqués sur des boucles imbriquées ou sur des équations récurrentes. Dans ce rapport, on analyse les systèmes d'équations récurrentes linéaires, une généralisation des équations récurrentes uniformes. La première partie de ce rapport décrit une méthode qui permet de trouver automatiquement si un tel système peut être ordonné à l'aide d'une fonction de temps affine, indépendamment du paramètre de taille de l'algorithme. Dans la seconde partie, on décrit une méthode puissante qui permet de transformer des équations récurrentes linéaires en équations récurrentes uniformes. Les résultats de ces deux parties reposent sur les propriétés des ensembles de points entiers appartenant à des polyèdres convexes. Nos résultats sont illustrés sur l'algorithme délimination de Gauss et sur l'algorithme de diagonalisation de Gauss-Jordan.

Abstract The parallelization of many algorithms can be obtained using space-time transformations which are applied on nested do-loops or on recurrence equations. In this paper, we analyze systems of linear recurrence equations, a generalization of uniform recurrence equations. The first part of the paper describes a method for finding automatically whether such a system can be scheduled by an affine timing function, independent of the size parameter of the algorithm. In the second part, we describe a powerful method that makes it possible to transform linear recurrences into uniform recurrence equations. Both parts rely on results on integral convex polyhedra. Our results are illustrated on the Gauss elimination algorithm and on the Gauss-Jordan diagonalization algorithm.

*This work was partially funded by the French Coordinated Research Program C^3 and by a Grant from the SOREP company

†IRISA, Campus de Beaulieu, 35042 RENNES-CEDEX, FRANCE
‡PRLB, Av. Van Becelaere, No. 2, Box 8, 1170 Brussels, Belgium

CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE
(L. A. 227)
UNIVERSITÉ DE RENNES 1 I. N. S. A. DE RENNES

INSTITUT NATIONAL DE RECHERCHE
EN INFORMATIQUE ET EN AUTOMATIQUE
(LABORATOIRE DE RENNES)

1 Introduction

Designing efficient algorithms for parallel architectures is one of the main difficulties of the current research in computer science. As the architecture of super-computers evolves towards massive parallelism, it becomes necessary to design smart compilers that not only look for efficient vectorized programs, but also try to optimize the distribution of the algorithm among the processors. In order to obtain good performance, the distributed algorithm must minimize as much as possible the amount of communications between remote processors, and balance the computational and the communication power of the processing units. One very promising methodology involves describing the algorithm using abstract specifications such as recurrence equations, as it was suggested by Karp et al. [17] in 1967. The algorithm to be mapped is specified as a set of equations attached to integral points, and mapped using a regular scheduling and allocation scheme on the architecture. A similar approach was used later by Lamport [21] for the parallelization of do-loops, and became the basis of many studies on the synthesis of systolic arrays [1, 2, 6, 20, 23, 24, 25, 29, 32]. The main problems that were tackled were the scheduling of the computations, the mapping of the computations on regular architectures, partitioning schemes, and optimal organization of multistep algorithms.

In this paper, we address the analysis and mapping of linear recurrence equations on parallel architectures. In section 2, we recall informally the principles of recurrence mapping, on the example of the matrix multiplication. Section 3 is devoted to a formal definition and presentation of linear recurrences and algorithms. After defining a "normal form" of recurrence equations, we give a constructive necessary and sufficient condition for the existence of a linear timing function for such equations. This result extends the previously known results on uniform recurrence equations, and makes it possible to relax the hypothesis that the target architecture is only locally connected. Our results are illustrated by the analysis of Gauss and Gauss-Jordan algorithms. Section 4 tackles the problem of transforming linear recurrence equations into uniform recurrence equations. Such a transformation is mandatory when the target architecture is locally connected, for example, a systolic array.

2 Principle of recurrence mapping

Consider the multiplication $C = AB$ of $N \times N$ matrices. Denote $a(i,j)$, $b(i,j)$, and $c(i,j)$ the elements of matrices A, B and C respectively. We can use the following system of recurrence equations to describe this algorithm :

$$1 \leq i \leq N, 1 \leq j \leq N, 1 \leq k \leq N \quad \rightarrow \quad C(i,j,k) = C(i,j,k-1) + a(i,k) \times b(k,j) \tag{1}$$

$$1 \leq i \leq N, 1 \leq j \leq N, k = 0 \quad \rightarrow \quad C(i,j,k) = 0 \tag{2}$$

$$1 \leq i \leq N, 1 \leq j \leq N \quad \rightarrow \quad c(i,j) = C(i,j,N). \tag{3}$$

Each one of these equations represents a set of unique assignement statements attached to an integral point of the space. Equations (1) and (2) are attached to integral points of a cube of size N in $\mathbf{Z}^3$, and define an iterative calculation of the expression

$\sum_{k=1}^{N} a(i,k)b(k,j)$, using an intermediate variable C. Equation (3) has indexes in $\mathbf{Z}^2$, and simply "renames" the results of the recurrence as c. Of course, many other recurrence forms would give the same result, and the choice which is made here, although reasonable, is arbitrary.

The mapping procedure, as it was described in [25] or [30], amounts to perform an index transformation, usually referred to as *space-time transformation*, that affects each statement to an instant of time and a processor location. Such a transformation must satisfy basically two conditions :

- first, if a statement depends on the result of another statement, it should be executed after the result itself is available,
- secondly, two statements allocated to the same processor should not be scheduled at the same instant of time.

If we restrict ourselves to the case when the index transformation is *linear*, it appears that the above conditions can be equivalently expressed as a linear programming problem, when one considers uniform recurrences. Linear transformations have proved to be very effective in order to design locally connected architectures such as systolic arrays. These are the main reasons why research in this domain has mainly focused on uniform recurrences, and linear transformations.

Before discussing the limitations of uniform recurrences, let us recall informally the principles of the mapping procedure, in the case of uniform recurrence equations, on the matrix multiplication example. The following development is based on results that appeared in many papers, among which [25, 30].

First, we transform the equations in order to obtain uniform recurrences. Equation (1) is not uniform, as the indexes of variables a and b are not translations of (i,j,k). In this particular case, the index mapping is a *projection*, and corresponds to the *broadcasting* of data a and b. Using a well known technique called *pipelining*, we can replace equation (1) by :

$$1 \le i \le N, 1 \le j \le N, 1 \le k \le N \rightarrow C(i,j,k) = C(i,j,k-1) + A(i,j,k) \times B(i,j,k) \quad (4)$$

$$1 \le i \le N, 1 \le j \le N, 1 \le k \le N \rightarrow A(i,j,k) = A(i,j-1,k) \quad (5)$$

$$1 \le i \le N, j = 0, 1 \le k \le N \rightarrow A(i,j,k) = a(i,k) \quad (6)$$

$$1 \le i \le N, 1 \le j \le N, 1 \le k \le N \rightarrow B(i,j,k) = B(i-1,j,k) \quad (7)$$

$$i = 0, 1 \le j \le N, 1 \le k \le N \rightarrow B(i,j,k) = b(k,j). \quad (8)$$

The detail of this transformation will be explained in section 4.4. The new system is composed of equations (2) to (8). Equations (4), (5) and (7) are said to be uniform, since the index functions of all the variables are translations of (i,j,k). The other equations of the system are not uniform, but as far as the timing analysis is concerned, they can be ignored, because they only serve defining inputs (equations (2), (6), (8)) or outputs of the system (3). The *dependence graph* that corresponds to this system is depicted in figure 1a for $N = 3$. The nodes of this graph are the points of the cube of $\mathbf{Z}^3$ of size 3, and edges represent dependences between the variables.

The next step of the mapping procedure is to find an affine schedule for the system, that is to say, a mapping

$$t : \mathbf{V} \times \mathbf{Z}^3 \longrightarrow \mathbf{Z} \tag{9}$$

$$(V,(i,j,k)) \longrightarrow t(V,z) = \lambda_1 i + \lambda_2 j + \lambda_3 k + \alpha_V \tag{10}$$

where $\mathbf{V}$ denotes the set of variables of the system. The mapping t must be such that if computation of variable $V(i,j,k)$ depends on variable $V'(i',j',k')$, then $t(V,(i,j,k)) \geq t(V',(i',j',k')) + 1$, assuming that the execution of any statement takes exactly one step of time. However, as the equations are uniform, this condition can be replaced by a finite set of linear inequalities. Indeed, as $C(i,j,k)$ depends on $C(i,j,k-1)$, we must have :

$$t(C,(i,j,k)) \geq t(C,(i,j,k-1)) + 1$$

which gives

$$\lambda_3 \geq 1$$

Applying the same treatment to the other dependences, we obtain :

$$\begin{array}{rl} \alpha_C - \alpha_A \geq 1; \alpha_C - \alpha_B \geq 1 & \text{from equation (4)} \\ \lambda_2 \geq 1 & \text{from equation (5)} \\ \lambda_1 \geq 1 & \text{from equation (7).} \end{array}$$

A solution to this system of linear constraints is $\lambda_1 = \lambda_2 = \lambda_3 = 1$, $\alpha_A = \alpha_B = 0$, and $\alpha_C = 1$. Therefore, a valid schedule for the system is that $C(i,j,k)$ be computed at time $i+j+k+1$, and $A(i,j,k)$ and $B(i,j,k)$ transmitted at time $i+j+k$.

The final step of the mapping procedure is to allocate the computations on a locally connected architecture. Basically, the method amounts to find a linear mapping a from $\mathbf{Z}^n$ to $\mathbf{Z}^{n-1}$, called the *processor allocation function*, such that $a(z)$ is the number of the processor that executes the calculations attached to point z. The mapping a must be chosen in such a way that a processor has no more than one calculation to perform at a given instant. If a is chosen to be a *projection* of the space, along a direction defined by a vector u, the only constraint on the choice of u is that u must not be parallel to the planes $i+j+k = T$ defined by the timing-function. In other words, $\lambda_1 u_1 + \lambda_2 u_2 + \lambda_3 u_3$ must be non zero.

In the case of the matrix multiplication, the projection of the equations along the direction $(0,0,1)$ produces the systolic architecture shown in figure 1b.

The interconnection pattern of the architecture is readily obtained by looking at the way the edges of the dependence graph are projected. Here, the elements of A and B enter the matrix, in a skewed fashion, respectively from the left and from the bottom of the array, and the partial results remain in the cells. Other architectures can be derived. For example, by projecting the dependence graph along $(1,1,1)$, we obtain an array, similar to the well-known systolic array first proposed by Kung and Leiserson ([19]), where both data and partial results are moving. Moreover, the control of the cells of the systolic array is simple and can be derived automatically. The only difficulty is that a given cell

may have to perform different calculations at different instant of time, as the equations are defined on subdomains. However, in his thesis, Rao ([32]) shows how one can replace conditions involving linear combinations of indexes by new boolean variables which are tested by the cells in order to decide which equation is to be applied. We will not develop furthermore this step (see [25, 26, 30] for example).

Uniform recurrence equations present several limitations which call for an extension, at least to linear recurrences, as we shall see in the next section :

- it is recognized that specifying an algorithm using uniform recurrences is often a difficult, and at least, a tedious task. This task becomes much simpler if linear recurrences are allowed;
- using uniform recurrences is somehow mandatory when one wants to implement an algorithm on a locally connected architecture, such as a systolic array. The reason is that it is not possible to implement non local communications in one instant of time. However, a lot of parallel architecture provide communication facilities via richer topologies : for example, hypercubes, shuffle-exchange networks, broadcast by bus etc. Therefore, it is tempting to consider more complex recurrences, such as linear ones (other index functions would also be interesting).
- In some cases, it is very difficult to replace linear dependences by uniform ones, especially when the calculations done by the algorithm depend recursively on its results. This is for example the case in the recursive convolution, or in the dynamic programming algorithm. Most of the recent work on the subject tries to overcome the problem, more or less formally. Therefore, it is interesting to try to solve this difficulty in its full generality.

In the following section, we study the scheduling of linear recurrences. For the sake of illustrating the paper, we will use, when necessary, results on allocation functions that were presented informally in this section.

3 Linear recurrence equations and algorithms

3.1 Statement of the problem

Definition 1 A *linear parameterized recurrence equation* is an equation of the form

$$z \in D_p \rightarrow U(z) \quad = \quad f[V(I(z)), ...] \tag{11}$$

where :

- z is a point of $\mathbf{Z}^n$, where $\mathbf{Z}$ denotes the set of integers,
- $p = (p_1, p_2, ..., p_m)$ is a point of $\mathbf{Z}^m$ named *the size parameter* of the equation. We assume that p belongs to a convex polyhedron $\mathbf{P} \subset \mathbf{Z}^m$ (in most cases, $\mathbf{P} = \mathbf{N}^m$, where $\mathbf{N}$ denotes the set of non negative integers).

- D_p is the set of integral points belonging to a convex polyhedron of $\mathbf{Z}^n$, called the *domain* of the equation[1]. We assume that D_p is bounded[2] and is defined by a finite set of linear inequalities involving z and p.

- I is an affine mapping from $\mathbf{Z}^n$ to $\mathbf{Z}^l$ called *index mapping*; I has the form

$$I(z) = A.z + B.p + C$$

  where the constants A, B and C are integral matrices, $A \in \mathbf{Z}^l \times \mathbf{Z}^n$, $B \in \mathbf{Z}^l \times \mathbf{Z}^m$, and $C \in \mathbf{Z}^l$,

- U and V are *variable names* belonging to a finite set $\mathbf{V}$. Each variable is indexed with an integral index, whose dimension (called the *index dimension* of the variable in the following) is constant for a given variable. The variable $U(z)$ is called the *result* of the equation and $V(I(z))$ is an *argument.*

- f is a single-valued function that depends *strictly* on its arguments; we assume that the function f has complexity $O(1)$.

- the '...' means that there can be other arguments of the same form as $V(I(z))$.

- the domains of two equations having the same variable as result are disjoints. This hypothesis ensures that a variable is not defined twice.

For a given p, equation (11) represents a finite set of *equation instances*, each one of which is associated with a particular point z of D_p.

A *system of linear recurrence equations* is a finite set of equations such as (11), having the same parameter set. Note that all equations need not be indexed in the same subspace, i.e., n is not necessarily the same for all equations.

We call *variable instance of the system* any term $U(t)$ that appears in an equation instance. The *index domain* of a variable is the set of indexes of the instances of the variable. Note that the index domain of a variable is not necessarily a convex polyhedron, but is however a finite union of convex polyhedra. A variable instance that appears only in the left-hand side of an equation is called an *output.* Similarly, a variable instance that appears only in the right-hand side of an equation is called an *input.* Variable instances that are neither outputs nor inputs are called *intermediate data.* Variables whose instances are all inputs (respectively, outputs or intermediate data) are called *input variables* (respectively, *output variables* or *intermediate variables*).

A particular case of linear recurrence equations is when the index mapping reduces simply to a translation. The equation is then said to be *uniform.* The importance of uniform recurrence equations for expressing parallel computations was first noticed by Karp et al. [17]. Our definition of uniform equations is however less restrictive that the

[1]The reader is referred to [35] for an introduction to convex polyhedra.

[2]This hypothesis is not compulsory, but simplifies the following presentation. In particular, it can be useful to assume that the domain has one infinite direction, in order to represent algorithms such as the convolution. All the results in this paper can be extended to cover such a case.

definition in [17] which makes the assumption that all the equations of a system have the *same* domain.

3.2 Normal form of a system of equations

The purpose of this subsection is to show that one can replace a system of linear recurrence equations by a new equivalent system, which is more convenient for the analysis of the dependences. The goal is to separate clearly the inputs, the outputs and the intermediate data, since only the dependences between intermediate data have to be considered for studying the schedule of the equations. Moreover, it is necessary that all equations be attached to integral points of the same space.

We say that the argument of an equation is *fully indexed*, if its index dimension is the same as the index dimension of the result of the equation. An equation is *fully indexed*, if all its arguments are fully indexed. For example,

$$1 \leq i \leq n, 1 \leq j \leq n \rightarrow U(i,j) = V(i,j-1)$$

is fully indexed, but

$$1 \leq i \leq n, 1 \leq j \leq n \rightarrow U(i,j) = V(i)$$

is not, because $V(i)$ is not fully indexed.

An equation is an *input equation*, if f is the identity function, and the only argument of the equation is an input variable. Similarly, an equation is an *output equation*, if f is the identity function, and U is an output variable. Finally, an equation is a *computation equation*, if its result and arguments are all intermediate variables

Definition 2 A system of linear recurrence equations is said to be in *normal form*, iff :

- all the variables are either input, output or intermediate variables,
- all equations are either input, output or computation equations,
- all the computation equations are fully indexed, and have the same index dimension.

The following theorem holds:

Theorem 1 *For any system of linear recurrence equations, there exists an equivalent system which is in normal form.*

A formal proof of the theorem can be found in [37]. Basically, it involves applying successively three transformations to the initial system of equations. Rather than describing formally these transformations, we illustrate them on examples.

Example 1 The first transformation, called *variable normalization*, modifies the system in such a way that only input, output or intermediate variables remain. To illustrate this, consider the equation :

$$1 \leq i \leq N \rightarrow A(i) = A(i-1).$$

The variable A has one input instance $A(0)$, one output instance $A(N)$, and intermediate instances $A(i)$, when $1 \leq i \leq N-1$. Therefore, A is neither an input, an output or an intermediate variable. We can rewrite this equation in the following way:

$$\begin{aligned} 2 \leq i \leq N-1 &\rightarrow A_1(i) = A_1(i-1) \\ i = 1 &\rightarrow A_1(i) = A_2(0) \\ i = N &\rightarrow A_3(i) = A_1(i-1). \end{aligned}$$

The domain of A is partitioned into three new domains, each one of which corresponding respectively to the intermediate, input and output instances of A, renamed respectively A_1, A_2, and A_3. □

Example 2 The purpose of the second transformation, called *input and output separation*, is to make input (respectively output) variables appear only in input (respectively output) equations. Consider for example the system

$$\begin{aligned} 1 \leq i \leq N &\rightarrow A(i) = A(i-1) + B(3i) \qquad (12) \\ i = 0 &\rightarrow A(i) = x(i) \\ i = N &\rightarrow C(i) = A(i). \end{aligned}$$

In this system, the input variable B appears in equation (12) which is not an input equation. However, the property holds when equation (12) is replaced by

$$\begin{aligned} 1 \leq i \leq N &\rightarrow A(i) = A(i-1) + B'(i) \\ 1 \leq i \leq N &\rightarrow B'(i) = B(3i). \end{aligned}$$

□

Example 3 Finally, the last transformation, called *full indexing*, aims at obtaining a fully indexed system. Consider the following system of equations, which is a somewhat simplified version of the dynamic programming algorithm :

$$\begin{aligned} 1 \leq i < j \leq N, i < k < j &\rightarrow C(i,j,k) = f[C(i,j,k+1), c(i,k)] \qquad (13) \\ 1 \leq i < j \leq N, k = j &\rightarrow C(i,j,k) = w(i,j) \qquad (14) \\ 1 \leq i < j+1 \leq N &\rightarrow c(i,j) = C(i,j,i+1) \qquad (15) \\ 1 \leq i \leq N, j = i+1 &\rightarrow c(i,j) = w(i,j). \qquad (16) \end{aligned}$$

A straightforward method to obtain fully indexed computation equations is to change the index dimension of c, by adding a 0 as a third component. We then obtain the new

system :

$$\begin{aligned}
1 \leq i < j \leq N, i < k < j &\rightarrow C(i,j,k) = f[C(i,j,k+1), c(i,k,0)] \\
1 \leq i < j \leq N, k = j &\rightarrow C(i,j,k) = w(i,j) \\
1 \leq i < j+1 \leq N, k = 0 &\rightarrow c(i,j,k) = C(i,j,i+1) \\
1 \leq i \leq N, j = i+1, k = 0 &\rightarrow c(i,j,k) = w(i,j).
\end{aligned}$$

However, this transformation is far from being optimal, in the sense that it amounts to "place" variables $c(i,j)$ arbitrarily in the index space, which may have consequences on the schedule of the equations.

In this particular example, a much better method is to substitute $c(i,k)$ in (13) by its definition. We obtain the new system :

$$\begin{aligned}
1 \leq i < j \leq N, i < k+1 < j &\rightarrow C(i,j,k) = f[C(i,j,k+1), C(i,k,k+1)] &(17)\\
1 \leq i < j \leq N, i = k+1 < j &\rightarrow C(i,j,k) = f[C(i,j,k+1), w(i,k)] &(18)\\
1 \leq i < j \leq n, k = j &\rightarrow C(i,j,k) = 0 &(19)\\
1 \leq i < j+1 \leq N &\rightarrow c(i,j) = C(i,j,i+1) &(20)\\
1 \leq i \leq N, j = i+1 &\rightarrow c(i,j) = w(i,j) &(21)
\end{aligned}$$

where (17) is fully indexed.

The substitution method cannot always be used, in particular when the substituted variable depends on itself. In this case, the substitution creates a number of equations that depends on the parameter p. The problem of finding an index transformation which merges optimally several index spaces is therefore still open. □

In the remaining of this paper, we will consider normal form systems. We define the *domain* $\mathcal{D}$ of a (normal form) system of equations as the convex hull of the union of the domains of its computation equations (input and output equation domains excluded).

3.3 Dependence vectors

Assume that $V(I(z))$ is an intermediate variable instance. Then, $V(I(z))$ is defined by another equation, attached to point $I(z)$. In a normal form system, the intermediate variable instances are fully indexed, and both z and $I(z)$ belong to $\mathbf{Z}^n$. Hence the vector $z - I(z)$ is defined and is called *dependence vector*. Dependences with results or data are not considered, as they do not correspond to effective computations. Given an argument $V(I(z))$ of an equation, we denote by $\Theta_{V(I(z))}$ (or simply Θ when there is no ambiguity) the dependence vector $z - I(z)$. More formally, $\Theta_{V(I(z))}$ is a function from $\mathbf{Z}^{n+m}$ to $\mathbf{Z}^n$ which maps a pair (z,p) to $z - I(z)$. In the following, we shall denote by $Range(\Theta_{V(I(z))})$ the range of $\Theta_{V(I(z))}$ when $p \in \mathbf{P}$ and $z \in D_p$.

The following proposition is a key result of our study :

Proposition 3.1 *$Range(\Theta_{V(I(z))})$ is a convex polyhedron of $\mathbf{Z}^n$.*

Proof As I is linear in z and p, the dependence vector is an affine mapping :

$$\begin{aligned}
J : \mathbf{Z}^{n+m} &\rightarrow \mathbf{Z}^n \\
(z,p) &\rightarrow z - I(z) = (\mathcal{I} - A)z - Bp - C
\end{aligned}$$

where $\mathcal{I}$ is the identity matrix. Clearly, J is affine. Moreover, for all p, D_p is a convex polyhedron, which depends linearly on p, and p itself belongs to a convex polyhedron of $\mathbf{Z}^m$. Therefore, the set $\{(z,p) \mid p \in \mathbf{P}, z \in D_p\}$ is a convex polyhedron of $\mathbf{Z}^{n+m}$. It results that $Range(\Theta_{V(I(z))})$, being the image of a convex polyhedron by an affine mapping, is also a convex polyhedron of Z^n. ■

As a consequence of proposition 3.1, any dependence vector $\Theta_{V(I(z))}$ can be expressed as the sum of a convex combination of the vertices of the convex set $Range(\Theta_{V(I(z))})$, of a positive combination of its extremal rays, and of a linear combination of its lines[3].

3.4 Computability and scheduling

The scheduling problem is to find a function that associates each variable instance $U(z)$ with a given instant of time t, in such a way that the arguments needed for the calculation of $U(z)$ are already calculated at time t. If such a mapping exists, the system is said to be *explicit* or *computable*. This is not always the case, as shown by the following equations :

$$\begin{aligned} A(i) = 1 \leq i \leq n &\rightarrow A(i-1) + B(i-1) \\ B(i) = 1 \leq i \leq n &\rightarrow A(i+1). \end{aligned}$$

Here, $A(i)$ depends on itself.

The property of computability has been investigated by Karp et al. [17], Rao [32], Yaacoby and Capello [43], and Delosme and Ipsen [6]. Karp et al. have shown that this property can be checked, when all the equations are uniform, have the same (possibly infinite) domain, and use only *strict* functions. These assumptions exclude the case when equations are defined on subdomains, as described here. In his thesis, Rao [32] gives a procedure to check the computability of recurrence equations defined on different domains, but his procedure is based on the fact that all equations are extended to the same domain.

Depending on the assumptions that are made, different results which are summarized here can be obtained :

- if the domains of the equations are bounded, and not parameterized, it is obvious that this property can be checked by testing whether the dependence graph is acyclic or not.
- if the domains of the equations are not bounded, and all equations are defined on the same domain, and the function f are strict, then the computability can be checked (result of Karp et al.).
- concerning equations defined on unbounded domains, when the domains are not the same for all equations[4], it has been shown by Joinnault [12] that the computability

[3]Recall that a vertex of a convex polyhedron is an extremal point of the polyhedron, a ray is the direction of an infinite half-line of the polyhedron, and a line is the direction of a line of the polyhedron. Any convex polyhedron can be generated by a finite set of vertices, of rays and of lines. Again, the reader is referred to [35] for definitions.

[4]This is the assumption we have made in this paper.

of a system of uniform recurrence equation is undecidable. The proof amounts to show that any Turing machine can be encoded as a uniform recurrence system of equations and to show that the computability of uniform recurrence equations reduces to the termination of a Turing machine.

- an interesting question is to find out whether a *parameterized* system of recurrence equation is computable. In other words, when the domains of the equations depend on an integral size parameter, is it possible to check the computability of all instances of the system ? Recently, Saouter and Quinton ([34]) have shown that this property is also undecidable.

The last result shows that it is hopeless in general to find out if there exists an ordering of the calculations compatible with the dependences. However, as we shall see in the next section, results can be obtained in the special case of a *linear* ordering, which is of practical interest.

3.5 Linear timing functions

A particular case of ordering on the calculations is called a *affine timing function* [25, 29]. If such a function can be found, of course, the system is computable. As we shall see, we can determine automatically whether there exists such a timing-function, all at once for a parameterized system of equations.

Consider a variable instance $U(z)$, and define for each variable a function $t(U,z)$ from $\mathbf{V} \times \mathbf{Z}^n$ to $\mathbf{Z}$ of the form :

$$t(U,z) = \lambda_1 z_1 + ... + \lambda_n z_n + \alpha_U$$

where λ_1, ..., λ_n are integers independent of U. Denote $\lambda^t z = \lambda_1 z_1 + ... + \lambda_n z_n$.

Assume that the evaluation of each function of the system takes at least one unit of time. The following result gives a constructive means for obtaining the function t :

Theorem 2 *The number* λ_1, ... , λ_n, $\alpha_U, U \in \mathbf{V}$ *define a timing function for all* p *iff :*

- *(i) for all vertex* σ *of all the dependence sets Range*$(\Theta_{V(I(z))})$ $\lambda^t \sigma + \alpha_U - \alpha_V > 0$,
- *(ii) for all extremal ray* ρ *of the dependence sets Range*$(\Theta_{V(I(z))})$, $\lambda^t \rho \geq 0$
- *(iii) for all line* ν *of the dependence sets Range*$(\Theta_{V(I(z))})$, $\lambda^t \nu = 0$.

Proof

- $\boxed{\Rightarrow}$ We have to prove that for all p and all $z \in D_p$, $t(U,z) - t(V,I(z)) \geq 1$, i.e.

$$\lambda^t (z - I(z)) + \alpha_U - \alpha_V \leq 1$$

As $\theta = z - I(z)$ belongs to $Range(\Theta_{V(I(z))})$, it can be decomposed using the vertices, rays and lines of $Range(\Theta_{V(I(z))})$ as

$$\theta = \sum \alpha\sigma + \sum \beta\rho + \sum \delta\nu$$

Using the hypotheses, we obtain :

$$t(U,z) - t(V,I(z)) \geq \lambda^t \sum \alpha\sigma + \alpha_U - \alpha_V$$

and as $\sum \alpha = 1$, we have

$$\begin{aligned} t(U,z) - t(V,I(z)) &\geq \sum \alpha[\lambda^t\sigma + \alpha_U - \alpha_V] \\ &\geq 1. \end{aligned}$$

- [⇐] Conversely, assume that $\lambda_1, \ldots, \lambda_n, \alpha_U, U \in \mathbf{V}$ define a timing function for all p. Consider a vertex σ of $Range(\Theta_{V(I(z))})$. There exist p and $z \in D_p$ and an equation

$$z \in D_p \rightarrow U(z) = f[V(I(z)), \ldots]$$

such that $z - I(z) = \sigma$. Therefore we must have :

$$\lambda^t\sigma + \alpha_U - \alpha_V \leq 1$$

which proves *(i)*.

Let ρ be the ray of some $Range(\Theta_{V(I(z))})$. Assuming that *(ii)* is not true, it is simple to show that there exists an equation such that $t(U,z) < t(V,I(z))$. A similar reasoning applies for *(iii)*. ■

Note that Theorem (2) is a generalization of a result given by Rajopadhye and Fujimoto [31] to the case of paremeterized recurrence equations. A similar condition was also presented, independently of ours, by Irigoin and Triolet [16], in the framework of Nested Loops analysis.

3.6 Gaussian elimination and Gauss-Jordan diagonalization

In the following, we shall use the Gaussian elimination, and the Gauss-Jordan diagonalization as examples. We first present the Gauss-Jordan algorithm, from which the Gaussian elimination can be deduced immediately by removing one equation.

Let A be a $N \times N$ matrix, and b be a $N \times 1$ vector. The problem is to solve the linear system $Ax = b$, by the Gauss-Jordan elimination algorithm. For the sake of commodity, we assume that A is a $N \times (N+1)$ matrix whose last column is b. Elements of A are denoted $a(i,j)$. The basic step of the algorithm is as follows: at step k, element $a(k,k)$ is taken as the pivot, and is used to zero out elements $a(i,k)$, $1 \leq i \leq N$, $i \neq k$. At the end of the algorithm, i.e. when $k = N$, matrix A is the identity matrix, and b is the solution x of the system. Let $A(i,j,k)$ denotes the value of element (i,j) of matrix A at step k. The algorithm can be precisely specified by the following equations :

Input Equations :

$$k = 0, 1 \leq i \leq N, 1 \leq j \leq N \quad \rightarrow \quad A(i,j,k) = a(i,j) \tag{22}$$
$$k = 0, 1 \leq i \leq N, j = N+1 \quad \rightarrow \quad A(i,j,k) = b(i) \tag{23}$$

Computation Equations :

$$1 \leq k \leq N, k \leq j \leq N+1, i = k \quad \rightarrow \quad A(i,j,k) = A(i,j,k-1)/A(k,k,k-1) \tag{24}$$
$$1 \leq k \leq N, k \leq j \leq N+1, 1 \leq i < k \quad \rightarrow \quad A(i,j,k) = A(i,j,k-1) - A(i,k,k-1)/A(k,k,k-1) \times A(k,j,k-1) \tag{25}$$
$$1 \leq k \leq N, k \leq j \leq N+1, k < i \leq N \quad \rightarrow \quad A(i,j,k) = A(i,j,k-1) - A(i,k,k-1)/A(k,k,k-1) \times A(k,j,k-1) \tag{26}$$

Output Equations :

$$1 \leq i \leq N \quad \rightarrow \quad x(i) = A(i, N+1, N) \tag{27}$$

It can be readily seen that the system is in normal form. The Gauss-Jordan domain is shown in figure 2 and is :

$$\mathcal{D} = \{(i,j,k) \mid 1 \leq i \leq N, k \leq j \leq N+1, 1 \leq k \leq N\}.$$

The Gaussian elimination differs from the Gauss-Jordan diagonalization in that elements above the pivot are not zeroed. Therefore, equation (25) has to be removed. At the end of the algorithm, matrix A is upper triangular, and it remains to perform a back substitution in order to solve the system.

Consider the Gauss-Jordan algorithm. Table 3 summarizes the dependences of the algorithm. Let us apply Theorem 2. Consider for example the dependence vector $(0, j-k, 1)$ from equation (24). When N ranges over $\mathbf{N}$, this vector belongs to the half-line Θ whose origin is $(0,0,1)$ and direction is $(0,1,0)$. In other words, Θ is a convex polyhedron with vertex $(0,0,1)$ and (extremal) ray $(0,1,0)$. Therefore, we must have :

$$\lambda_3 \geq 1 \text{ and } \lambda_1 \geq 0.$$

Applying the same analysis to the other dependence vector gives the final set of constraints :

$$\lambda_1 \geq 0, \lambda_1 \leq 0, \lambda_2 \geq 0, \lambda_3 \geq 1.$$

Therefore, a possible timing function is $t(A,(i,j,k)) = k$. Notice that $t(A,(i,j,k)) = j+k$ is also valid, but the coefficient of i has to be 0. This is because the domain for the possible values of λ contains the line $j = k = 0$.

On the other hand, if we analyze the Gaussian elimination algorithm, the constraints are

$$\lambda_1 \geq 0, \lambda_2 \geq 0, \lambda_3 \geq 1$$

and $t(A(i,j,k)) = i+j+k$ is now a valid schedule, as i is not constrained to be 0. This comes directly from the fact that equation (25) has been removed, and therefore, the constraint $\lambda_1 \leq 0$ is not necessary.

4 Uniformization

Unless it is uniform, a system of recurrence equations cannot be mapped directly on a systolic array. Indeed, as the dependence vectors are not constants, processors may have to communicate with an arbitrary large number of other processors, and this is not possible in a systolic array, where the processors are only connected to their neighborhood.

This section is concerned with *uniformization*, that is to say, the transformation of linear recurrences into uniform ones. The uniformization problem has not yet received a full answer. It is tackled by Fortes and Moldovan [11], Wong and Delosme [42], Kuhn [18], Gachet et al. [13], Van Dongen and Quinton [36], Joinnault [12], Rajopadhye [31], and others (see section 5). We begin by explaining informally the principle of our method (subsection 4.1). Then we describe precisely the uniformization method (subsection 4.2). In subsection 4.3, we sketch how the method can be applied in the case of the Gauss-Jordan algorithm.

4.1 Informal description of the uniformization method

Let us briefly illustrate the uniformization process with the Gaussian elimination algorithm. Consider the non-uniform dependence $(i-k,0,1)$ between $A(i,j,k)$ and $A(k,j,k-1)$ in equation (26) (figure 4). As explained in section 3.6, the associated domain $\Theta_{V(I(z))}$ is infinite; it has one vertex $(1,0,1)$ and a ray $(1,0,0)$. To solve the uniformization problem, we express the dependence vector as a non-negative integral linear combination of these vectors, called *uniformization vectors.* A solution is:

$$(i-k,0,1) = (i-k-1).(1,0,0) + (1,0,1). \tag{28}$$

The uniformization method involves rewriting the variable instance $A(k,j,k-1)$ using a new variable A', which is defined by a (uniform) equation of the form $A'(i,j,k) = A'(i-1,j,k)$, where the dependence vector is $(1,0,0)$. The same transformation is repeated for all vectors of the decomposition, until the non-uniformity is removed. In our example, this transformation gives the uniform dependence graph of figure 4b.

The main difficulty of the method is to choose the uniformization vectors in such a way that the resulting uniform system of equations has a linear timing-function. As we shall see, the choice of the uniformization vectors depends on several factors, among which the dimension of the domain of the initial equation, the dimension of its image by the index mapping I, and the value of the vectors of the decomposition.

4.2 Automatic synthesis of the uniformization vectors

Given a set of linear recurrences in its normal form, we only consider here the uniformization of dependences resulting from the computation equations. The case of input equations will be solved in section 4.4.

The organisation of this section is the following. In subsection 4.2.1, we consider the case when pipelining vectors are needed, that is to say, when the null space of I and

the linear space generated by the domain D of the equation where I appears have a non empty intersection. Then, in subsection 4.2.2, we consider the case when no pipelining is needed. Finally, we summarize the method in subsection 4.2.3.

4.2.1 Pipelining vectors

Consider an equation :

$$z \in D \rightarrow U(z) = f(..., V(I(z)), ...). \tag{29}$$

Definition 3 Given a convex polyhedron D, we note $Vect(D)$ the linear space parallel to D, that is to say the direction of the affine hull of D[5]. We say that a vector A is *parallel* to D iff $A \in Vect(D)$. We note $dim(D)$ the dimension of $Vect(D)$. Given an affine mapping I, we note $Null(I)$ the null space of the linear part of I. □

In the remaining of this section, we will assume that the system of linear recurrence equations where equation (29) appears has a timing-function, and moreover that the ranges of all dependence vectors do not contain the vector 0 and are enclosed in a pointed cone, named Θ^*. This excludes in particular the case when the range of one of the dependence vector contains a line.

Definition 4 A vector $\phi \neq 0$ is said to be a *pipelining vector* iff $\phi \in Null(I) \cap Vect(D)$.

Proposition 4.1 *If ϕ is a pipelining vector, then the following system is equivalent to equation (29):*

$$z \in D \rightarrow U(z) = f(..., V'(z), ...) \tag{30}$$
$$z \in D_1 \rightarrow V'(z) = V'(z - \phi) \tag{31}$$
$$z \in D_2 \rightarrow V'(z) = V(I(z)) \tag{32}$$

where V' is a new variable and

$$D_1 = \{z \in D \mid z - \phi \in D\}$$
$$D_2 = D \setminus D_1.$$

Moreover, the domain D_2 can be partitioned into a finite number of convex polyhedra of dimension $dim(D) - 1$, and for all point $z \in D_2$, $z - I(z)$ belongs to the dependence set $\Theta_{V(I(z))}$ of equation (29).

[5]Given a set D, the affine hull of D is the smallest affine space which contains D. The direction of an affine space A is the unique linear space L such that A is a translation of L[9].

Proof For all $z \in D$, by repeated use of (31), there exists $k \in \mathbf{N}$ such that:

$$U(z) = f(..., V'(z - k\phi), ...)$$

and $z - k\phi \in D_2$. Then, using (32),

$$U(z) = f(..., V(I(z - k\phi)), ...).$$

However, as ϕ is a pipelining vector, $I(z - k\phi) = I(z)$. Therefore,

$$U(z) = f(..., V(I(z)), ...)$$

which proves the equivalence between (29) and the above system of equations. Since $D_2 = \{z \in D \mid z - \phi \notin D\}$, it can be partitioned into a finite number (independent on p) of convex domains of dimension $dim(D) - 1$. Finally, as $D_2 \subset D$, the set $\{\Theta_{V(I(z))} | z \in D_2\}$ is included in $\{\Theta_{V(I(z))} \mid z \in D\}$. ■

Example 4 In the case of the Gaussian elimination, equation (28) gives an integral decomposition of the dependence vector $(i - k, 0, 1)$ on the vectors $(1, 0, 0)$ and $(1, 0, 1)$. In fact, vector $(1, 0, 0)$ belongs both to the null space of I and to $Vect(D)$ and is therefore a pipeline vector. The pipelining transformation, when applied to equation (26), yields:

$$1 \le k \le N, k \le j \le N+1, k < i \le N \rightarrow A(i,j,k) = A(i,j,k-1) - A(i,k,k-1)/A'(i,j,k) \times A(k,j,k-1) \quad (33)$$

$$1 \le k \le N, k \le j \le N+1, k+1 < i \le N \rightarrow A'(i,j,k) = A'(i-1,j,k) \quad (34)$$

$$1 \le k \le N, k \le j \le N+1, i = k+1 \rightarrow A'(i,j,k) = A(k,j,k-1). \quad (35)$$

One can check that the domain of equation (35) has dimension 2, one less than domain of (33). □

When $Null(I) \cap Vect(D) \neq \{0\}$, it is necessary to propagate the data along pipelining vectors first, using Proposition 4.1. However, the pipelining vectors do not always belong to Θ^*, as shown by the following example.

Example 5 Consider the following system of equations, whose dependence graph is shown in figure 5, for $p = 3$.

$$1 \le j \le p, i \ge j - 1, i \le 2j - 1 \rightarrow A(i,j) = B(j-1, 0).$$

In this case, the domain of the dependence vector is the cone whose extremal rays are $(1, 1)$ and $(0, 1)$. On the other hand the vector $(1, 0)$ is a pipelining vector, and does not belong to the cone generated by the dependence vectors. □

The following proposition shows that one can choose the pipelining vectors in such a way that the dependence vectors of the system after transformation still belong to a pointed cone. As a consequence, we will be guaranteed that a timing-function still exist after transformation.

Lemma 1 *Given a pointed cone C, and a linear space V, there exists a pointed cone C' which contains C and a basis of V.*

Proof Let $\{v_i\}_{1\leq q\leq n}$ be a basis of V. We shall prove that there exists a set of values $\{\epsilon_i = \pm 1\}_{1\leq q\leq n}$ such that $\{\epsilon_i v_i\}_{1\leq q\leq n}$ and C generate a pointed cone C'. Let $\{c_k\}_{1\leq k\leq K}$ be the set of extremal rays of C. Suppose first that the basis has one single vector v. Denote C_v the cone generated by C and v. If C_v is pointed, the problem is solved, with $\epsilon = 1$. Suppose that C_v is not pointed. Let us call *separating hyperplane of* C any hyperplane $h^t z = 0$ such that $h^t c_k > 0$ for all extremal ray c_k of C. As C is pointed, such an hyperplane exists. Moreover, as C_v is not pointed, for all separating hyperplane of C, $h^t v \leq 0$. If there exists such an hyperplane for which $h^t v < 0$, then $-h^t v > 0$, and C_{-v} is a pointed cone, and the proposition holds. If not, then $h^t v = -h^t v = 0$, and $-v \in C_v$. Therefore, $-v$ can be written as a positive combination of the extremal rays of C and of v, i.e:

$$-v = \sum_{k=1}^{K} \mu_k c_k + \nu v$$

where $\nu \geq 0$. A simple calculation shows then that

$$-v = \sum_{k=1}^{K} \frac{\mu_k c_k}{1+\nu}$$

and therefore, $-v$ is a positive combination of the extremal rays of C, which implies that $-v \in C$. Again, C_{-v} is a pointed cone. The case when $q > 1$ is solved by applying the same idea successively to all the vectors of the basis. ■

Using Lemma 1, we have directly the following proposition:

Proposition 4.2 *If Θ^* is a pointed cone, and if $Null(I) \cap Vect(D) \neq \{0\}$, there exists a basis of $Null(I) \cap Vect(D)$ which, together with Θ^*, generates a pointed cone.*

As a direct result of the above development, when $Null(I) \cap Vect(D) \neq \{0\}$, we can replace equation (29) by an equivalent system of equations, where none of the remaining non-uniformities involve pipelining vectors anymore, the dependence vectors still belong to Θ^*, and the dimensions of D and of $I(D)$ are the same.

4.2.2 Routing vectors

We now consider the case when no pipelining vector exist. In the following, we shall say that a dependence vector $\Theta_{V(I(z))}$ has an *integral decomposition* on a set of vectors $\{A_i\}_{1\leq i\leq q}$ in the domain D_p, if there exist q linear mappings $\alpha_i(z,p)$ such that for every $p \in \mathbf{P}$, and every $z \in D_p$

$$\Theta_{V(I(z))} = \sum_{j=1}^{q} \alpha_j(z,p).A_j \tag{36}$$

and $\alpha_i(z,p) \in \mathbf{N}$. Vectors A_i will be called *routing vectors* in the following.

To obtain such a decomposition, the idea is to use the extremal rays of the cone Θ^* as uniformization vectors. The following result, which is due to Delsarte ([8]), ensures that we can always find and compute such an integral decomposition:

Proposition 4.3 *Given a rational convex polyhedral pointed cone C of dimension n, there always exist a pointed cone Γ whose extremal rays R_1, ... , R_n form a unimodular basis, and such that $C \subset \Gamma$.*

Proof We assume that the number of rays of C is $m \geq n$ (If $m < n$, one can transform the problem into one in $\mathbf{Z}^m$.) The polar cone C^0 of C, is the pointed cone defined as

$$C^0 = \{y \in \mathbf{Z}^n \mid x^t.y \leq 0, \forall x \in C\}.$$

We must find a pointed cone Γ^0 of extremal rays R_1^0, ... R_n^0 such that $\Gamma^0 \subset C^0$. Indeed, if such a cone exists, the polar cone Γ of Γ^0 solves the problem, as $C \subset \Gamma$, and the extremal rays R_1, ..., R_n of Γ are such that

$$(R_1, ..., R_n) = -((R_1^0, ..., R_n^0)^{-1})^t.$$

Let M be the non-singular matrix $(M_1, ..., M_n)$ of the extremal rays of C^0. It can be shown that there always exists a rational positive upper triangular matrix P such that

$$M.P = U \tag{37}$$

where U is unimodular. Because of (37) and because P is positive, the unimodular U defines the extremal rays of Γ^0, i.e., $U = (R_1^0, ..., R_n^0)$. ■

The direct consequence of Proposition 4.3 is that the pointed cone Θ^* which contains the ranges of all the dependence vectors can be enclosed in another pointed cone Γ, whose extremal rays form a *unimodular basis.* Therefore, all the dependence vectors have an integral decomposition on the rays of Γ, which is simply their (unique) linear decomposition on the vectors of the unimodular basis. A simple change of basis provides the linear mappings $\alpha_i(z, p)$ which are sought.

It remains to show that we can remove the non-uniformities of the system of equations, using elementary transformations involving the routing vectors. We consider successively two cases depending on whether the routing vector belong to $Vect(D)$ or not.

Proposition 4.4 *If A_i is a routing vector which does not belong to $Vect(D)$, there exists an affine mapping I' such that the following system of equations is equivalent to equation (29)*

$$z \in D \rightarrow U(z) = f(..., V'(z), ...) \tag{38}$$
$$z \in D_1 \rightarrow V'(z) = V'(z - A_i) \tag{39}$$
$$z \in D_2 \rightarrow V'(z) = V(I'(z)) \tag{40}$$

where V' is a new variable and

$$D_1 = \{z \in D \mid z \in D \wedge 0 \leq k < \alpha_i(z, p)\}$$
$$D_2 = \{z \in D \mid z \in D \wedge k = \alpha_i(z, p)\}.$$

Proof Clearly, if A_i is not parallel to D, then for all pair of points z and z' of D, and for all pair of integers k and k', $z - kA_i \neq z' - k'A_i$. Therefore, $D_1 \cap D_2 = \emptyset$, and each variable V' is defined only once. Moreover, the (linear) function which maps z to $z - \alpha_i(z,p)A_i$ is one-one, and there exists therefore an affine mapping J such that $z = J(z - \alpha_i(z,p)A_i)$. By taking $I'(z) = I(J(z))$ in equation (40), we obtain an equivalent system. ■

It remains to solve the case when a routing vector A is parallel to D. Before giving the solution, we show on an example why the above method fails.

Example 5 Figure 6 illustrates what happens when the routing vector is parallel to D. The equations are:

$$0 \leq i \leq p \;\rightarrow\; U(i) = U(i+p).$$

When p ranges over $\mathbf{N}$, the dependence vector $-p$ belongs to the half-line generated by -1. The transformation described by Proposition 4.4 gives the following system of equations:

$$\begin{aligned} 0 \leq i \leq p &\;\rightarrow\; U(i) = U'(i) \\ 0 \leq i \leq 2p-1 &\;\rightarrow\; U'(i) = U'(i+1) \\ p+1 \leq i \leq 2p &\;\rightarrow\; U'(i) = U(i). \end{aligned}$$

It is clear that $U'(i)$ is defined twice for $p+1 \leq i \leq 2p-1$. □

To solve the problem, we will show that we can augment the pointed cone Θ^* so that A is replaced by two other vectors which, together with Θ^*, generate a pointed cone.

Proposition 4.5 *If A is a routing vector which belongs to $Vect(D)$, and if $dim(D) < n$, then there exist vectors u and v such that:*

- *neither u nor v belong to $Vect(D)$*
- *Θ^*, u and v generate a pointed cone.*

Proof We assume that A is an extremal ray of Θ^* (If it is not the case, we can beforehand replace A by a positive combination of extremal rays of Θ^*). As we have supposed that there are no pipelining vectors, $dim(D) = dim(I(D)) < n$. Therefore, the supplementary linear space S of $Vect(D) \cap Vect(I(D))$ is not reduced to 0. Let v be a non zero vector of S. By Lemma 1, either Θ^* and v or Θ^* and $-v$ generate a pointed cone. Assume that this is true for v. We claim that Θ^*, v and $A - v$ still generate a pointed cone. Indeed, as Θ^* and v generate a pointed cone, and as v and A are not collinear, there exists a separating hyperplane $h^t z = 0$ of Θ^* such that $h^t A > h^t v$. Therefore, $h^t(A - v) > 0$, and $h^t z = 0$ is also a supporting hyperplane of the cone generated by Θ^*, v and $A - v$. The vectors v and $A - v$ are the new vectors we seek in order to replace A, as none of these vectors belongs to $Vect(D)$. ■

The consequence of Proposition 4.5 is that one can replace a routing vector by two new vectors, in such a way that the transformation of Proposition 4.4 is possible, and that the new system still has a timing-function. The following example illustrates this situation.

Example 6 Consider the following system of equations (see figure 7):

$$i = 0, 0 \leq j \leq n, 0 \leq k \leq n \quad \rightarrow \quad A(i,j,k) = A'(k, j+n, 0). \tag{41}$$

In this example, the cone Θ^* has the extremal rays $(0,1,0)$ and $(1,1,-1)$. The dependence vector is $(-k,-n,k) = (-n-k)(0,1,0) - k(1,1,-1)$. However, as the first ray is parallel to D, the transformation using the routing vector $(0,1,0)$ is incorrect. One can avoid the problem by replacing the first vector by a combination of the second one and the first one. Here, we rewrite $(0,1,0) = (1,1,-1) + (-1,0,1)$. The new routing (see figure 7b) is done by $(-k,-n,k) = -(n+k)(-1,0,1) - (n+2k)(1,1,-1)$. □

4.2.3 Summary of the Uniformization method

In summary, the uniformization method works as follows:

- First case: $Null(I) \cap Vect(D) = \{0\}$.
  - Assume first that $dim(D) < n$. We do not need pipeline vectors. Using Proposition 4.3, we find out a set of at most n vectors A_i which form a unimodular basis generating a pointed cone enclosing Θ^*. Then, we process successively all the vectors A_i. If A_i is not parallel to D, then we apply the transformation defined in Proposition 4.4, which gives a new domain D. If A_i is parallel to D, using Proposition 4.5, we replace A_i by two new vectors, in such a way that these vectors still generate a pointed cone.
  - If $dim(D) = n$, our method does not make it possible to obtain a uniform system in $\mathbf{Z}^n$. However, it is always possible to reindex the variables in $\mathbf{Z}^{n+1}$, and to apply the above theory. Indeed, the new system will need more processors, as the allocation function will map the system to $\mathbf{Z}^n$ instead of $\mathbf{Z}^{n-1}$. This is however not surprising: the case when $dim(D) = dim(I(D)) = n$ represents a bad situation when the dependence vectors are "dense" in $\mathbf{Z}^n$.
- If $Null(I) \cap Vect(D) \neq \{0\}$, then we can find an integral basis $\Phi = \{\phi_i\}_{1 \leq i \leq q \leq n}$ of $Null(I) \cap Vect(D)$ such that Φ and Θ^* still generate a pointed cone Θ^+. Then, by applying Proposition 4.1 successively to each vector of Φ, we eventually obtain a finite set of equations, whose domain have dimension $dim(D) - q$, and whose dependence vectors are still in Θ^+. We are back to the previous case.

All this discussion is summarized by the following theorem:

Theorem 3 *Given a set of linear parameterized recurrence equations whose dependence vectors are non null and belong to a pointed cone, there exists an equivalent system of uniform recurrence equations which has a timing-function.*

It should be noted that although the above transformations involve solving some problems for which no efficient algorithms exist (such as for example, the computation of the generating system of a convex polyhedral domain), our experience has shown that in practice, the systems we handle lead to small size problems which can be solved in a reasonable amount of time.

We also emphasize that our method has no pretention to be optimal. Clearly, at several steps, some choices may result in more or less efficient transformation. At least, however, we know that the resulting system can be scheduled.

4.3 Multistep algorithms

The method described in the preceding section applies when Θ^* is pointed. We illustrate here what may be done when Θ^* is not pointed.

Consider the uniformization of the dependence $(i-k, 0, 1)$ (as above) in the Gauss-Jordan elimination algorithm. The set of dependences is defined by the line $k = 1$ (see 3.6). Hence, Θ^* is a cone defined by $k \geq 0, j = 0$ (a half-plane). This cone, shown in figure 8, is degenerated since it contains the line $k = 0$; one may verify that $(1,0,0)$, $(-1,0,0)$ and $(0,0,1)$ form an integral unimodular basis for this cone. The resulting uniform dependence graph is shown on figure 9. However, after uniformization, this algorithm has no timing-function, as there are opposite dependence vectors.

We now present an *ad-hoc* method for solving this problem. The opposite vectors were used in the uniformization of $A(k,k,k-1)$ and of $A(k,j,k-1)$. By looking at the recurrence equations, one observes that these two partial results are computed in the part of the domain where $i-k \geq 0$. Hence, the Gauss-Jordan algorithm can be considered as a two-step algorithm. In the first step, when $i-k \geq 0$, both $A(k,k,k-1)$ and $A(k,j,k-1)$ are computed. In the second one, both variables are used only as inputs.

Both steps can be uniformized independently with the above uniformization method. Then, the two steps can be put together as described in [6] and [40]. By looking at the time cones of the two algorithms, one can compute a reindexing transformation which gives a maximal global time cone. Finally, the translation of the second step can be computed by expressing the fact that all external dependences must be positive along the time. Figure 10 shows the resulting uniform dependence graph; the two steps of the algorithm are shown; one of them was translated from i to $i+N$.

This uniform dependence graph being defined, one can now map it on various systolic arrays. Let us choose for example the mapping $a(i,j,k) = (i,k)$, i.e., a projection along axis j. This allocation function is authorized, as the direction $(0,1,0)$ of the projection is not orthogonal to the vector λ, therefore ensuring that two computations allocated to the same processor are not executed at the same instant of time (see [29]). The resulting architecture is shown in figure 11, and was described independently by Robert and Trystram [33] and Lewis and Kung [22][6]. The movement of the A coefficients is the following : they enter the architecture in the bottom row, then they are reflected by the

[6]Both papers deal with the Algebraic Path Problem, which covers as a particular case the Gauss-Jordan elimination.

left diagonal, move rightwards to the right diagonal, where there are reflected again to the top. This movement is the projection of the path of the coefficients in the domain of figure 10.

4.4 Uniformization of the input equations

We now consider the uniformization of the *input equations*. The difference with what we saw previously is that there is a priori no location associated to the data: initially, a data can be anywhere. It can be considered as the transformation of data broadcasting into pipelining. Such a problem has been tackled already by several authors [11, 12, 31, 42]. We hereafter briefly present a method that uses some of the theory developed before. The way we solve the problem is illustrated on the input equation

$$A(i,j,k) = a(i,k) \tag{42}$$

of the matrix multiplication algorithm. The first step consists in defining the set of point (i,j,k) that will use a particular data $a(i_0,k_0)$. This is done by computing the kernel of the affine transformation that maps (i,j,k) on (i_0,k_0), and by intersecting this kernel with the domain of the index points (i,j,k). The kernel is the solution of the system

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} i \\ j \\ k \end{pmatrix} = \begin{pmatrix} i_0 \\ k_0 \end{pmatrix}. \tag{43}$$

Hence, the kernel is $(i,j,k) = (i_0,j_0,k_0) + k.(0,1,0)$. The intersection with the domain of the indices gives

$$(i,j,k) = (i_0,0,k_0) + k.(0,1,0) \text{ with } k \in [1,N]. \tag{44}$$

The resulting domain is a convex polyhedron, whose vertices are $(i_0,1,k_0)$ and (i_0,N,k_0). The second step is to choose an initial location in this convex, which defines where the data is first. In the systolic model, this location must be a vertex of the domain (otherwise, there would be no affine schedule). On the two possibilities, we arbitrary chose $(i_0,1,k_0)$. Equation (42) can now be rewritten, with the use of a new variable U_a, as

$$A(i_0,j,k_0) = U_a(i_0,1,k_0) \tag{45}$$

$$U_a(i_0,1,k_0) = a(i_0,k_0). \tag{46}$$

Equation (45) being fully indexed, it can be transformed into a set of uniform recurrences with the method described before. Equation (46) is a new input equation. Yet, its kernel only contains one point; hence, nothing needs to be broadcasted and uniformized.

5 Related work

In [43], Yaacoby and Capello are interested by the scheduling of affine recurrences. They give a sufficient condition for an affine recurrence to be computable, a necessary and

sufficient condition for the existence of an affine schedule and a constructive means for finding the affine schedule.

In [27], Mongenet and Perrin present a methodology for the mapping of inductive problems on systolic arrays. In these problems, routing need to be performed for the partial results that are further used as data. Work in this direction was recently made by Clauss [5]. In their approach, they decompose non-uniform dependences into two sets of vectors, namely the generating vectors (in our terminology, the pipelining vectors) and the dependency vectors (some of which are the routing vectors). These vectors are automatically computed by looking at the domain of the dependences. Their approach, however, is restricted to inductive problems, a sub-class of linear recurrences. Also, because they do not consider the size parameters in the construction of their dependence domain, their resulting architecture may be non-modular.

In [10], Guerra and Melhem consider the synthesis of another class of affine recurrences, where the non-uniform dependences are of the form $\vartheta = (a_1, ..., a_{t-1}, i - j, a_{t+1}, ..., a_n)$, a_i are constants, and i and j are indices of the problem. Their synthesis is in two steps. First, they determine a coarse timing function by analyzing the set of dependence vectors (note that size parameters are not taken into account). In the second step, they perform a time-space index transformation, compatible with the coarse timing. They illustrate their methodology on the dynamic programming algorithm.

In [3], Chen presents a methodology for the derivation of systolic algorithms and architectures in a unified framework based on the language Crystal. First, the naive algorithm is transformed into a bounded order and bounded degree program (with bounded fan-in and fan-out degree). Then, it is further transformed by means of a space-time mapping. The method is illustrated with the dynamic programming algorithm. Clearly, the aim of fan-in and fan-out reductions is to uniformize the algorithm. The fan-in reduction is something we have not considered here. This problem occurs when a naive algorithm makes use of a census function (i.e. which are applied on a set of data). In a linear recurrence equation, this cannot be the case. The fan-out reduction is achieved by means of additional recurrence equations whose constant dependence vectors are, what we call, the uniformization vectors. For the dynamic programming algorithm, this reduction is quite simple.

In [11], Fortes and Moldovan discuss how and whether data broadcasts in an array processor with a given interconnection structure can either be eliminated or reduced by choosing an adequate linear schedule. They consider the class of linear recurrences whose variable instances $V(Az + B)$ are such that A is rank deficient. In order to have broadcast, the rank of A must be strictly less than n, the dimension of the index z. Also, the associated computations must be scheduled at the same time. For a given schedule, necessary conditions are given on the processor allocation so that no broadcast is required and the array interconnections support the necessary data communications. Finally, they discuss how linear schedules can explore the limited broadcasting capability offered by the array interconnections to implement large broadcasts. In the approach of Fortes and Moldovan, the interconnection structure of the array is fixed. In this paper, we do not consider this because the resulting array is more likely to be designed on a VLSI chip.

It would be useful for us to extend the presented methodology to the case where the interconnection structure is fixed. It seems that such a problem can be solved by adding additional constraints related to this structure, as explained in [11].

In [42], Wong and Delosme also consider the elimination of broadcasting. Hence, as in [11], they consider dependences whose index matrix is rank deficient. Similar to our approach, they are interested by the synthesis of systolic arrays and they do not make a priori restriction on its interconnection structure. Their problem is similar to ours, non-uniform dependences must be expressed as a linear combination of a finite set of uniformization vectors. Their approach is different in the sense that no affine schedule is required on the resulting dependence graph. Thus, dependences can be opposite as in the graph of figure 9. First, the canonical vectors are chosen as candidates for the uniformization. They give a necessary and sufficient condition for the acceptance of such a canonical propagation. Secondly, it is generalized to allow the inclusion of non-canonical vectors. They show that all broadcasts are transformable when using these general propagations. This approach is different since the schedule is not a priori taken into account. In the method that we present, the uniformization vectors are minimal in terms of the associated set of constraints on the affine schedule.

In [31], Rajopadhye and Fujimoto also consider the transformation of broadcasting into pipelining; hence, the index matrix A of the linear dependence is rank deficient. The pipelining vectors are also found by computing the null space of A. They show how to transform the linear recurrence into a set of conditional uniform recurrences, and how to propagate the control signals in the resulting architecture. They also illustrate their methodology on the dynamic programming algorithm (for optimal string parenthesization).

In [12], Joinnault considers both the pipelining problem and the routing problem. The first is solved by looking at the null space of the index matrix (as in [11, 31]). To solve the second problem, the non-uniform dependence is written in terms of n transvections and one translation. Yet, no systematic method is provided to find them.

In conclusion, several papers deal with the mapping of linear recurrences on regular arrays. Yet, they differ in the form of the linear recurrence equations, and on the requirements imposed on the uniformization vectors (existence or not of an affine schedule). Thus, comparing the results must be done with great care!

6 Conclusion

We have described the basics for the analysis of systems of linear recurrence equations and the principles of the mapping of such equations on parallel architectures. We also pointed out few problems that still need to be solved. Once overcome, it will be possible to implement such methods into software packages that will produce efficient code. Some of the techniques presented here are already being used in prototype tools for systolic array design ([14, 39]). The presented approach is very promising as it makes it possible to apply safe transformations on the recurrence equations that would otherwise be very difficult

to use. A complete methodology should probably include the results of several other methods, such as the ones presented in section 5. For example, a complete uniformization method should include automatic reindexing transformations as the folding operation, which can be used for some particular types of affine function.

The material that we have presented in this paper has some commonalities with the work that has been done on restructuring compilers for super-computers. Merging both fields will probably provide new fruitful research ideas.

Acknowledgements The authors would like to thank referee 3 for his very detailed and valuable comments. They are also greatly indebted to P. Delsarte who provided a proof of Proposition 4.3.

References

[1] P.R. Cappello and K. Steiglitz. "Digital signal processing applications of systolic algorithms," *VLSI Systems and Computations*, H.T.Kung, B. Sproull et G. Steel editors, Computer Science Press (1981), 245-254.

[2] M.C. Chen. Synthesizing systolic designs. In *1985 International Symposium on VLSI Technology, Systems and Applications, Taipei, Taiwan, R.O.C.*, 8-10 May 1985.

[3] M.C. Chen, Synthesizing VLSI architectures: Dynamic programming solver. In *Proc. of the 1986 International Conference on Parallel Processing*, pages 776–784, August 1986.

[4] C. Choffrut and K. Culik II, Folding of the plane and the design of systolic arrays. *Information Proc. Letters*, Vol. 17, pages 149–153, October 1983.

[5] P. Clauss, Contribution à la synthèse de réseaux systoliques. *Rapport de DEA*, Université de Besançon, 1987.

[6] J.M. Delosme and I.C.F. Ipsen. Efficient systolic arrays for the solution of Toeplitz systems: an illustration of a methodology for the construction of systolic architectures for VLSI. In W. Moore, A. McCabe, and R. Urquhart, editors, *International Workshop on Systolic Arrays*, pages 37–46, Adam Hilger, University of Oxford, UK. July 2-4 1986.

[7] J.M. Delosme and I.C.F. Ipsen. Systolic array synthesis : computability and time cones. In *Parallel Algorithms and Architectures*, pages 295–312, North-Holland. 1986.

[8] P. Delsarte, Private communication, february 1989.

[9] J. Dieudonné. *Algèbre linéaire et géométrie élémentaire*, Hermann, Paris. 1964.

[10] C. Guerra and R. Melhem, Synthesizing non-uniform systolic designs. In *Proc. of the 1986 International Conference on Parallel Processing*, pages 765–772. August 1986.

[11] J.A.B. Fortes and D.I. Moldovan. Data broadcasting in linearly scheduled array processors. In *Proc. 11th Annual Symp. on Computer Architecture*, pages 224–231, 1984.

[12] P. Gachet and B. Joinnault. Conception d'algorithmes et d'architectures systoliques. Thèse de l'Université de Rennes I, Sept 1987.

[13] P. Gachet, B. Joinnault, and P. Quinton. Synthesizing systolic arrays using DIASTOL. In W. Moore, A. McCabe, and R. Urquhart, editors, *International Workshop on Systolic Arrays*, pages 25–36, Adam Hilger, University of Oxford, UK, July 2-4 1986.

[14] P. Gachet, P. Quinton, C. Mauras and Y. Saouter. Alpha du Centaur : a Prototype Environment for the Design of Parallel Regular Algorithms. IRISA Research Report number 439, November 1988.

[15] F. Irigoin. Partitionnement de boucles imbriquées. une technique d'optimisation pour les programmes scientifiques. Thèse de l'Université Pierre et Marie Curie, Juin 1987.

[16] F. Irigoin and R. Triolet. Dependence approximation and global parallel code generation for nested loops. In M. Cosnard et al., editors, *Parallel and Distributed Algorithms*, North-Holland, 1989.

[17] R.M. Karp, R.E. Miller, and S. Winograd. The organization of computations for uniform recurrence equations. *Journal of the Association for Computing Machinery*, 14(3):563–590, July 1967.

[18] R. H. Kuhn. *Optimization and Interconnection Complexity for : Parallel Processors, Single-Stage Networks, and Decision Trees.* Technical Report UIUCDCS-R-80-1009, University of Illinois at Urbana-Champaign, February 1980.

[19] H.T. Kung. Let's design algorithms for VLSI systems. *Proc. of the Caltech conference on VLSI*, January 1979.

[20] S.Y. Kung. VLSI array processors. *IEEE ASSP Magazine*, 2(3):4–22, July 1985.

[21] L. Lamport. The parallel execution of Do-Loops. *Comm. ACM*, Vol. 17, N. 2, pp. 83–93, Feb. 1974.

[22] P.S. Lewis and S.Y. Kung. Dependence graph based design of systolic arrays for the algebraic path problem. In *20th Asilomar Conf.*, Nov. 10-12, 1986.

[23] G.H. Li and B.W. Wah. "The design of optimal systolic arrays," *IEEE Trans. Computers* 34, 1 (1985), 66-77.

[24] W.L. Miranker and A. Winkler. Space-time representations of systolic computational structures. *Computing*, 32:93–114, 1984.

[25] D.I. Moldovan. "On the analysis and synthesis of VLSI algorithms," *IEEE Trans. On Computers* C-31 (11), November 1982, pages 1121–1126.

[26] D.I. Moldovan and J.A.B. Fortes. Partitioning and mapping algorithms into fixed size systolic arrays. *IEEE Transaction on Computers*, C-35(1):1–12, January 1986.

[27] C. Mongenet and G.-R. Perrin, Synthesis of systolic arrays for inductive problems. In J.W. de Bakker, A.J. Nijman, and P.C. Treleaven, editors, *Parallel Architectures and Languages Europe*, pages 260–277, Springer-Verlag, June 1987.

[28] M. Newman, "Integral Matrices", Academic Press, 1972.

[29] P. Quinton. *The Systematic Design of Systolic Arrays.* Technical Report 193, Publication Interne IRISA, Avril 1983.

[30] P. Quinton, "Automatic synthesis of systolic arrays from uniform recurrent equations," *Proc. IEEE 11-th Int. Sym. on Computer Architecture*, Ann Arbor, MI, U.S.A., 1984, 208-214.

[31] S.V. Rajopadhye and R.M. Fujimoto. Systolic array synthesis by static analysis of program dependencies. In J.W. de Bakker, A.J. Nijman, and P.C. Treleaven, editors, *Parallel Architectures and Languages Europe*, pages 295–310, Springer-Verlag, June 1987.

[32] S.K.Rao, "Regular iterative algorithms and their implementations on processor arrays", *PhD Thesis*, Information Systems lab., Standford University, U.S.A., Octobre 1985.

[33] Y. Robert and D. Trystram. Systolic solution of the algebraic path problem. In W. Moore, A. McCabe, and R. Urquhart, editors, *International Workshop on Systolic Arrays*, pages 171–180, Adam Hilger, University of Oxford, UK, July 2-4 1986.

[34] Y. Saouter and P. Quinton. Computability of Recurrence Equations, IRISA Research Report, to appear, 1989.

[35] A. Schrijver. *Theory of Linear and Integer Programming. Wiley-Interscience series in Discrete Mathematics*, John Wiley and Sons, 1986.

[36] V. Van Dongen and P. Quinton. Uniformization of linear recurrence equations : a step towards the automatic synthesis of systolic arrays. In *International Conference on Systolic Arrays*, San Diego (USA), pages 473–482, May 1988.

[37] V. Van Dongen and P. Quinton. The Mapping of Linear Recurrence Equations on Regular Arrays, Manuscript M 264, Philips Research Lab. Brussels, October 1988.

[38] V. Van Dongen, *The transformation of N-dimensional linear recurrences into (N+1)-dimensional uniform recurrences.* Manuscript M 243, Philips Research Lab. Brussels, May 1988.

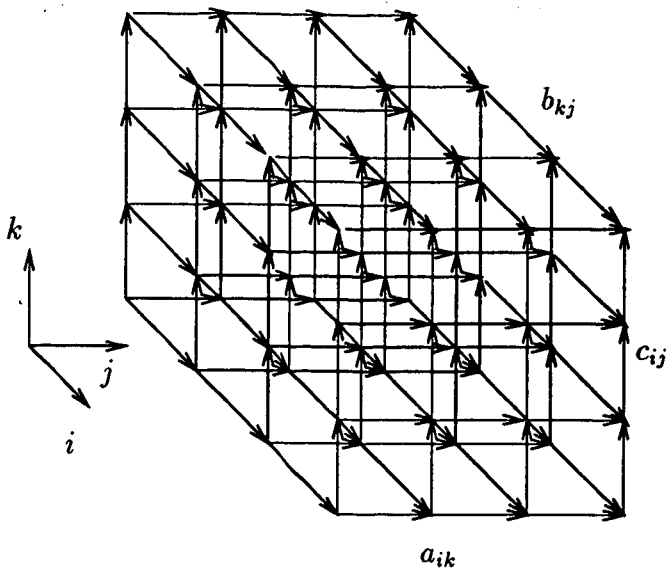
[39] V. Van Dongen. Presage, a Tool for the Design of Low-Cost Systolic Arrays. In *ISCAS 88*, pages 2765-2768, 1988.

[40] B.W. Wah, M. Aboelaze, and W. Shang. Systematic Designs of Buffers in Macropipelines of Systolic Arrays. *Journal of Parallel and Distributed Computing* 5, pp. 1-25, 1988.

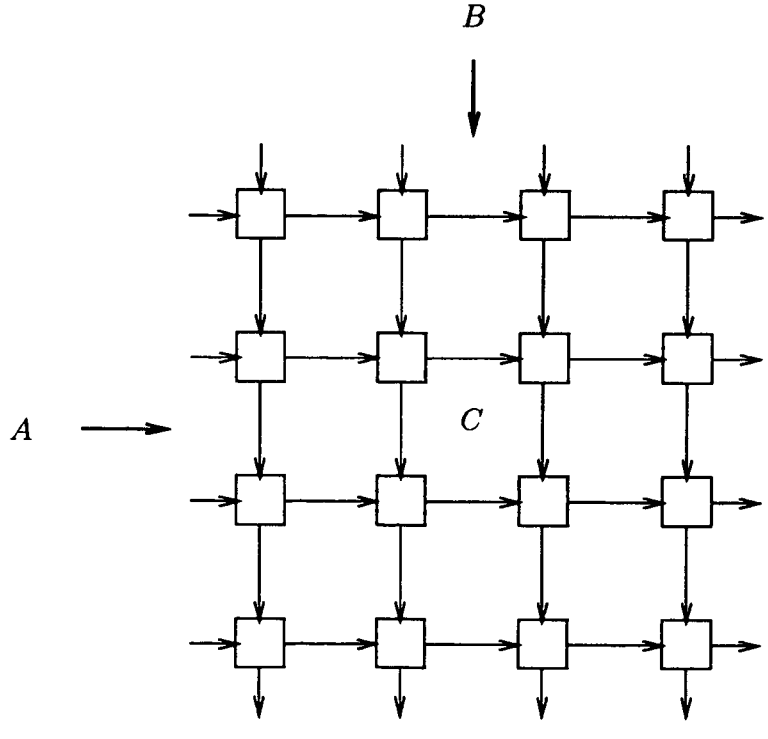
[41] Y. Wong and J.M. Delosme. Optimal systolic implementations of n-dimensional recurrences. In *IEEE Int. Conf. on Computer Design: VLSI in Computers*, pages 618–621, Oct. 7-10 1985.

[42] Y. Wong and J.M. Delosme. Broadcast removal in systolic algorithms. In *International Conference on Systolic Arrays*, San Diego (USA), pages 403–412, May 1988.

[43] Y. Yaacoby and R. Cappello, Scheduling a system of affine recurrence equations onto a systolic array. In *International Conference on Systolic Arrays*, San Diego (USA), pages 373–381, May 1988.



a)



b)

Figure 1: The matrix multiplication dependence graph, and the systolic array obtained by a projection along the k axis

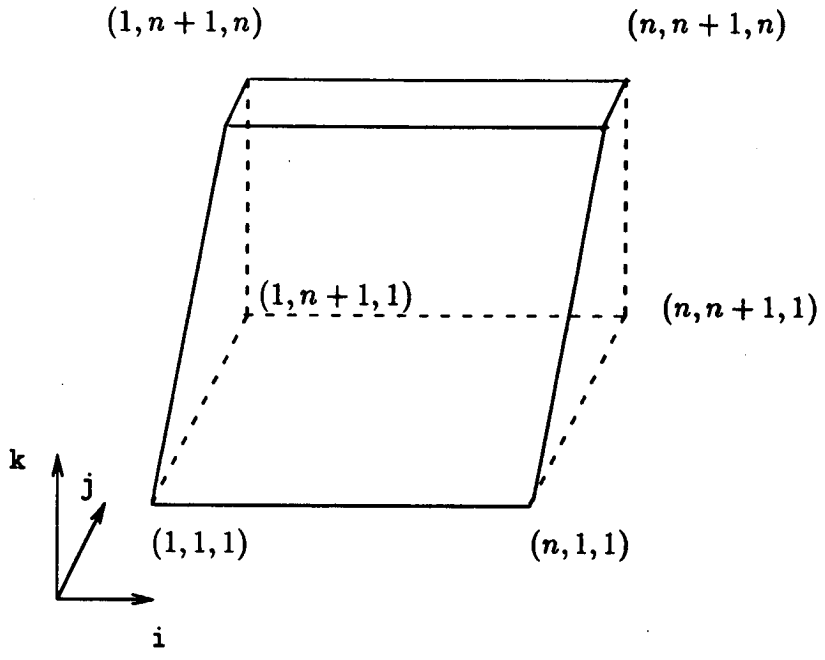


Figure 2: Domain of the Gauss-Jordan elimination algorithm

Equation	From	To	Vector	Note
(24)	$A(i,j,k)$	$A(i,j,k-1)$	$(0,0,1)$	
	$A(i,j,k)$	$A(k,k,k-1)$	$(0,j-k,1)$	$i=k, j \geq k$
(25)	$A(i,j,k)$	$A(i,j,k-1)$	$(0,0,1)$	
	$A(i,j,k)$	$A(i,k,k-1)$	$(0,j-k,1)$	$j \geq k$
	$A(i,j,k)$	$A(k,k,k-1)$	$(i-k,j-k,1)$	$i < k$
	$A(i,j,k)$	$A(k,j,k-1)$	$(i-k,0,1)$	$i < k$
(26)	$A(i,j,k)$	$A(i,j,k-1)$	$(0,0,1)$	
	$A(i,j,k)$	$A(i,k,k-1)$	$(0,j-k,1)$	$j \geq k$
	$A(i,j,k)$	$A(k,k,k-1)$	$(i-k,j-k,1)$	$i > k$
	$A(i,j,k)$	$A(k,j,k-1)$	$(i-k,0,1)$	$i > k$

Figure 3: Dependences in Jordan-Gauss elimination

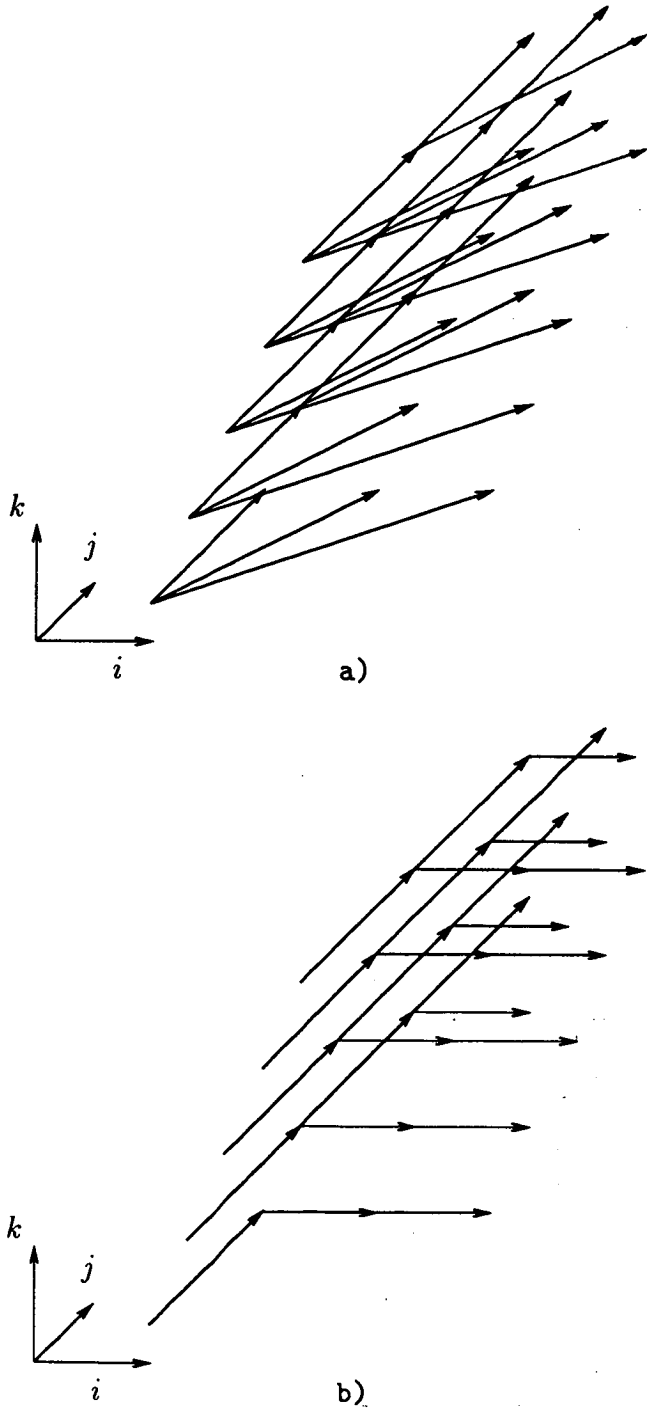


Figure 4: The dependence $(i-k, 0, 1)$ before and after uniformization

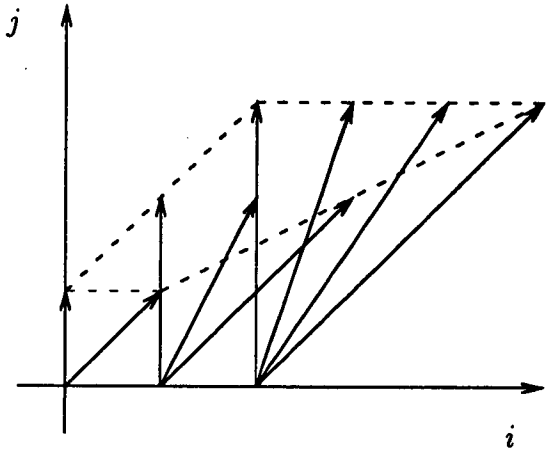


Figure 5: Case when a pipeline vector does not belong to Θ^*

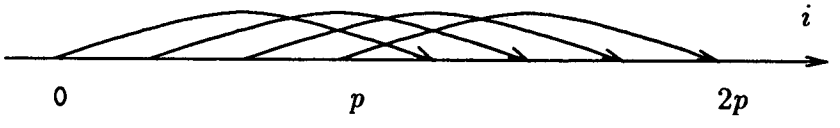


Figure 6: An example where the routing transformation is not valid

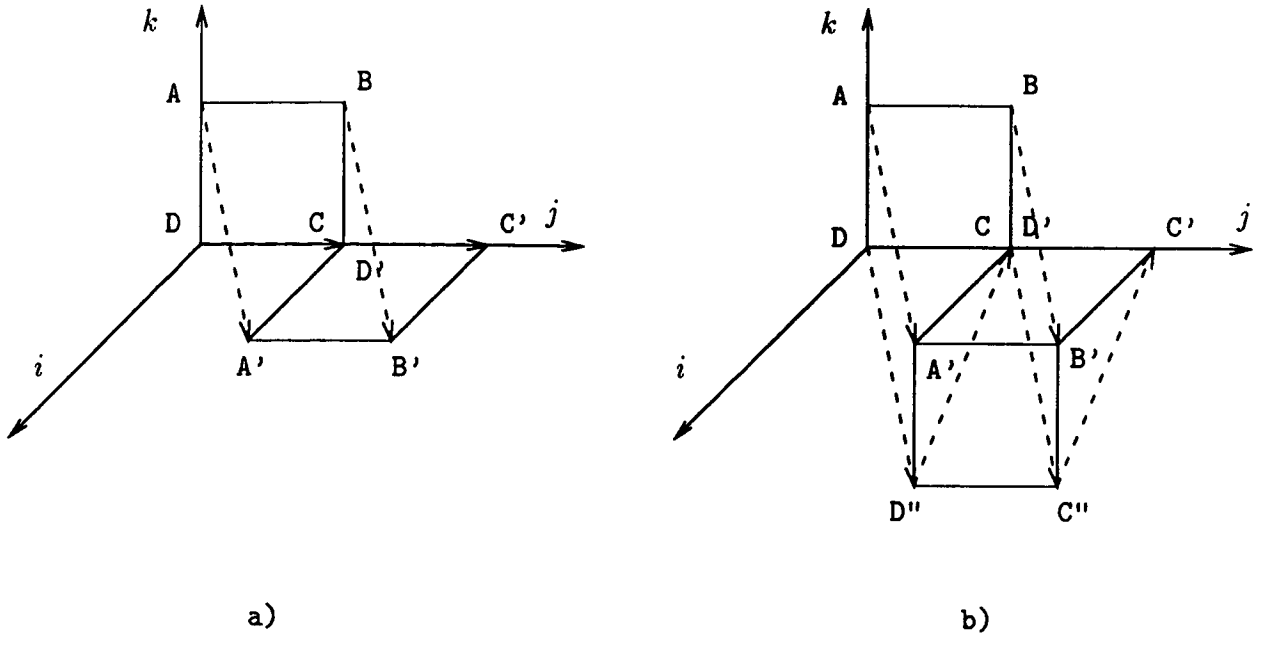


Figure 7: Case when a uniformization vector is parallel to D

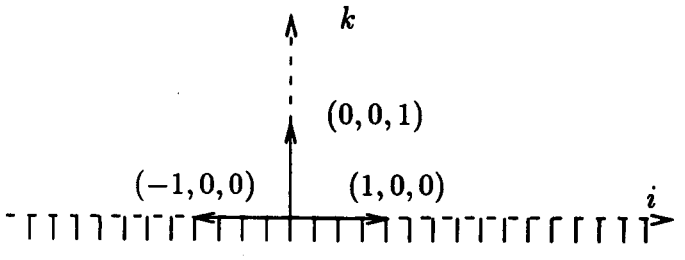


Figure 8: A degenerated cone Θ^* (it contains the line $k = 0$) and a possible minimal basis.

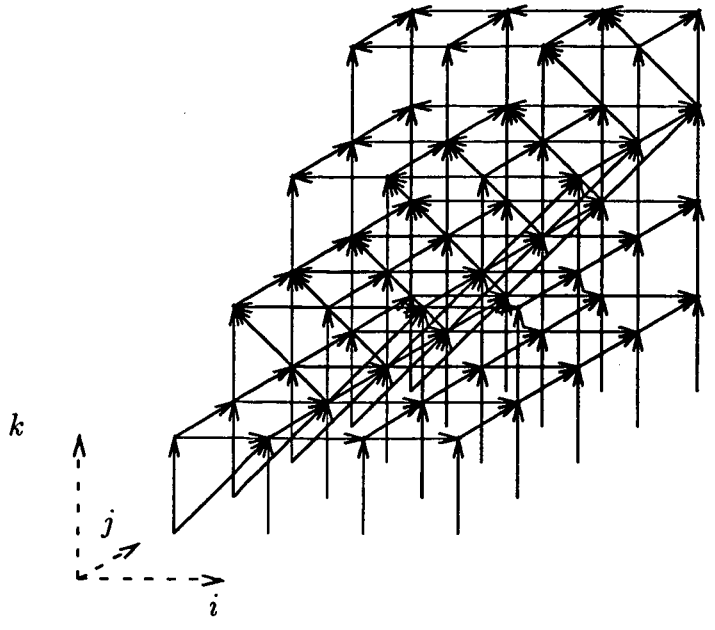


Figure 9: A uniform dependence graph for Gauss-Jordan which cannot be mapped on a systolic array.

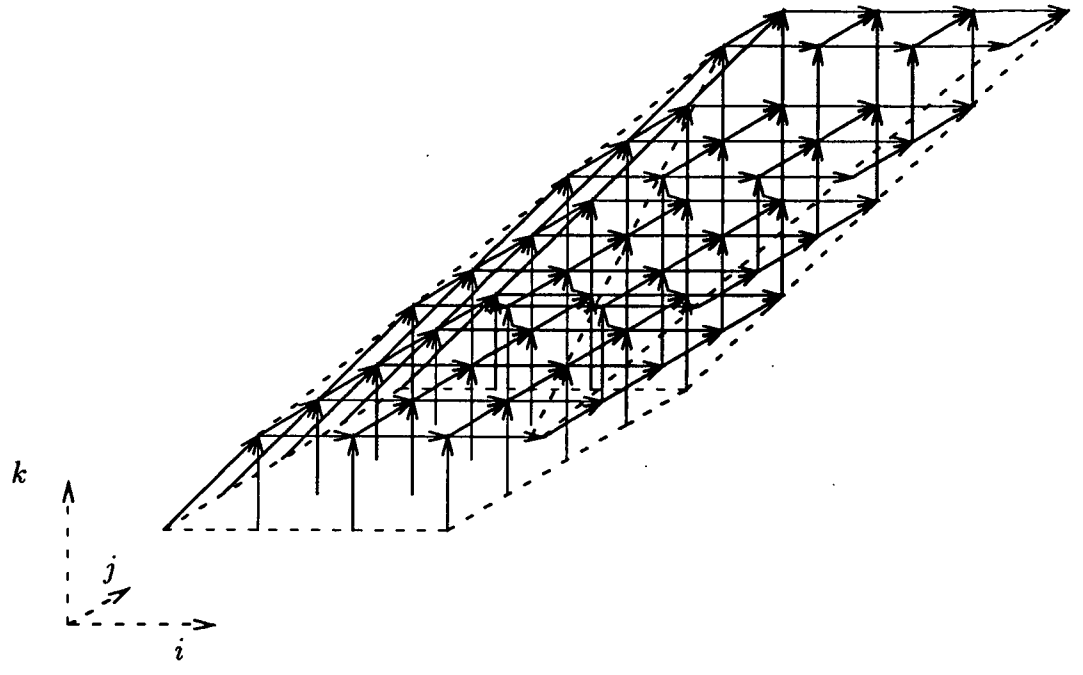
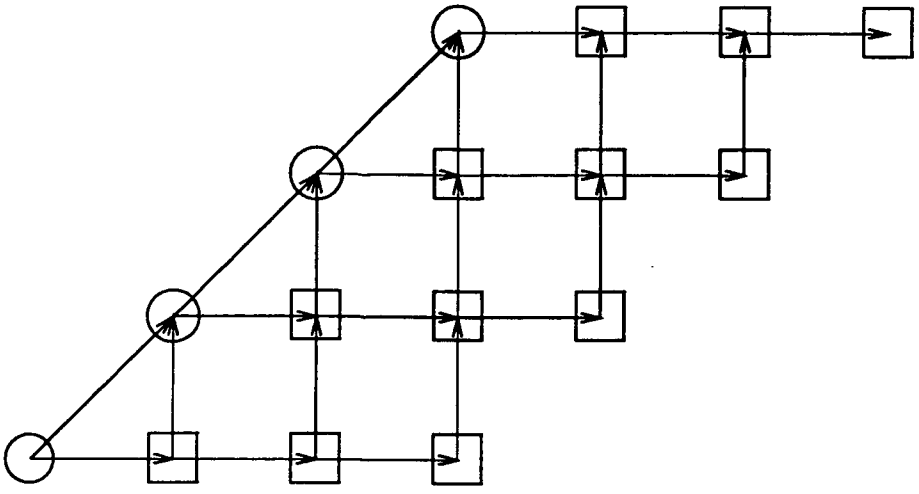


Figure 10: A uniform dependence graph for Gauss-Jordan, when considered as a two-step algorithm. The edges of the domains of each step are in dashed lines



a_{11}			
a_{12}	a_{21}		
a_{13}	a_{22}	a_{31}	
a_{14}	a_{23}	a_{32}	a_{41}
b_1	a_{24}	a_{33}	a_{42}
	b_2	a_{34}	a_{43}
		b_3	a_{44}
			b_4

Figure 11: Architecture obtained by projection along the j axis

PI 480 THE MODELLING SYSTEM PYRAMIDE AS AN INTERACTIVE HELP FOR THE GUIDANCE OF THE INSPECTION VEHICLE CENTAURE
Philippe EVEN, Lionel MARCE
22 Pages, Juin 1989.

PI 481 VERS UNE INTERPRETATION QUALITATIVE DE COMPORTEMENTS CINEMATIQUES DANS LA SCENE A PARTIR DU MOUVEMENT APPARENT
Edouard FRANCOIS, Patrick BOUTHEMY
40 Pages, Juin 1989.

PI 482 DEFINITION DE ALPHA : UN LANGAGE POUR LA PROGRAMMATION SYSTOLIQUE
Christophe MAURAS
18 Pages, Juin 1989.

PI 483 POLYNOMIAL IDEAL THEORETIC METHODS IN DISCRETE EVENT, AND HYBRID DYNAMICAL SYSTEMS
Michel LE BORGNE, Albert BENVENISTE, Paul LE GUERNIC
20 Pages, Juillet 1989.

PI 484 IMPLEMENTING ATOMIC RENDEZVOUS WITHIN A TRANSACTIONAL FRAMEWORK
Jean-Pierre BANATRE, Michel BANATRE, Christine MORIN
22 Pages, Juillet 1989.

PI 485 THE MAPPING OF LINEAR RECURRENCE EQUATIONS ON REGULAR ARRAYS
Patrice QUINTON, Vincent VAN DONGEN
40 Pages, Juillet 1989.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique